Class-Ordered LPA*: An Incremental-Search Algorithm for Weighted Colored Graphs

Jaein Lim Oren Salzman Panagiotis Tsiotras

Abstract— **Replanning is an essential problem for robots operating in a dynamic and complex environment, when the robot model of the environment changes continuously. Previous incremental-search algorithms efficiently reuse existing search results to facilitate a new plan when the environment changes. Yet, they rely solely on geometric information of the environment encoded in an edge-weighted graph. However, semantic information often provides valuable insights that cannot easily be captured quantitatively. We encode both semantic and geometric information of the environment in a weighted colored graph, in which the edges are partitioned into a finite set of ordered semantic classes (e.g., colors), and we incrementally search for the shortest path among the set of paths with minimal inclusion of inferior classes using information from the previous search using ideas similar to LPA*. The proposed Class-Ordered LPA* (COLPA*) algorithm inherits the strong theoretical properties of LPA*, namely, optimality and efficiency, but optimality is with respect to the total path order. Numerical examples show that semantic information helps reduce the relevant search space in a dynamic environment.**

I. Introduction

Replanning is a fundamental problem for robots operating in a complex and dynamic environment, where obtaining accurate models of the environment is difficult and the models themselves become quickly out of date [1]. Plans made on these models need to be quickly updated to resolve any relevant inconsistencies with the environment.

Consider, for example, a robot-navigation problem in a partially-known environment, where a robot having limited sensing plans for the shortest route. The robot must update its route when obstacles in the unknown region become known as the robot gathers more information about the environment [2]–[5]. Fast replanning is crucial for safe and efficient navigation while avoiding collisions with previously-unseen obstacles.

The significance of replanning becomes even more obvious when there are multiple robots navigating in the same environment. Unlike the previous example in which the obstacles are assumed to be static, multi-robot path finding problems involve dynamic obstacles induced by the robots' trajectories. Typically, more replanning is required to reach consensus for the team of robots to find a feasible path [6], [7].

J. Lim is a graduate student at the School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0150, USA, Email: jaeinlim126@gatech.edu

O. Salzman is an Assistent Professor at the Henry and Marilyn Taub Faculty of Computer Science, Technion, Haifa 3200003, Israel, Email: osalzman@cs.technion.ac.il

P. Tsiotras is the David and Andrew Lewis Professor at the Daniel Guggenheim School of Aerospace Engineering and the Institute for Robotics & Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332-0150, USA, Email: tsiotras@gatech.edu

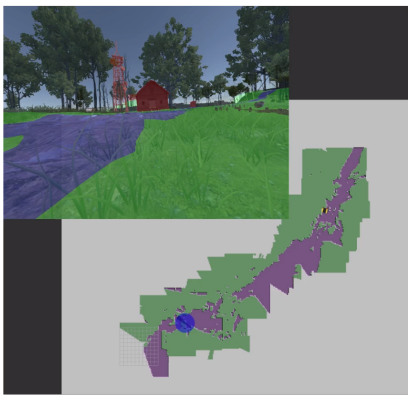


Fig. 1: Semantic segmentation of outdoor environment.

Replanning is necessary to resolve reoccurring inconsistencies of the models with the underlying uncertain environment. Incremental-search methods are well known techniques to quickly replan using previous results [8]–[10]. These methods use previous search results to facilitate a new plan by only replanning for the inconsistent part of the previous search, and hence they often show a significant speedup compared to searching from scratch, especially when the inconsistent part is small and close to the goal.

The previous incremental-search methods find efficiently a new plan based on previous search result when the environment is uncertain. However, they all rely on a fundamental assumption: the current information about the environment which identifies the "inconsistency" of the previous search is always accurate. Hence, the new plan found is optimal with respect to the most current information. However, this assumption may not always be true in uncertain environments where attaining accurate information is not always possible. For example, noisy sensors can contaminate the current information about a certain region of the environment, and hence the most-current information about that region may not be reliable. So, it would be prudent to plan a path that avoids these regions.

In addition, the environment often contains certain semantic information that is important for planning besides a binary classification of the environment such as known/unknown or reliable/unreliable. Consider, for example, an outdoor navigation problem where each part of the environment can be classified as road, grass, mud, and trees (see Figure 1). One may then want to find the shortest path with minimal

inclusion of trees, then mud, then grass, and then road. The question we address in this paper is how to replan in such environments, where both semantic and geometric information are imperative.

A. Related Work

A notion of optimality that incorporates semantic information was introduced by Wooden and Egerstedt [11] in the context of planning on a weighted colored graph, where the edges of the graph are partitioned into a finite set of classes with a total order, and the edges are also associated with numerical weights. They designed a new weight function based on the semantic (color) and geometric (weight) information of all the edges in the graph, so that the new weight function assigns to each edge a numerical cost. The shortest path with respect to the new weight is guaranteed to be the shortest path among the set of paths with minimal inclusion of edges of lower-quality classes [11]. For example, in the outdoor navigation problem in Figure 1 one may want to find the shortest path among the set of paths that includes the least number of edges that correspond to trees, then to mud, then grass, and then road. Once this weight map is found, and the original weights of all edges are modified accordingly, then a standard search algorithm such as Dijkstra [12] or A* [13] can find the optimal solution. However, to obtain the weight function, one needs to know the original weight and the color of all the edges in the weighted colored graph, which may not be possible for most applications.

In our previous work [14], we addressed this issue by generalizing A* to directly use the total order of paths defined in [11]. The Class-Ordered A* (COA*) in [14] finds the optimal path in a weighted colored graph, using an abstract queue to order the expansions of optimal path candidates. The COA* algorithm lazily and incrementally builds an optimal search tree, and hence only the edges along the optimal candidate path are evaluated (that is, the class and the weights of the edges are revealed). COA* is proven to be complete and correct, such that the algorithm terminates by finding the shortest path among the set of paths with minimal inclusion of inferior edges. However, COA* is a one-time planner which does not use previous search results. In this paper, we use the notion of optimality defined in COA* to incrementally search in a dynamic environment where both geometric and semantic information are important.

Incremental search methods are widely used technique to replan using geometric information of the environment. For example, when the environment is too complex, sequentially approximating the environment with samplings has been shown to be very effective for planning. Sampling-based planners densify the graph representation of the environment with incremental sampling, and then they plan sequentially on the sequence of graphs with increasing density to refine the solution [15]–[19]. The choice of replanning strategy dictates their convergence rate to the optimal solution.

Lifelong Planning A* (LPA*) is a well-known algorithm that uses a consistent heuristic to efficiently restrict replanning only to the relevant portion of the current search [9]. The inconsistent and relevant part of the search is repaired to find the new optimal solution. The provable efficiency of LPA* (that is, no vertices are expanded more than twice given a graph change) has made the LPA* algorithm the backbone for numerous applications where replanning is necessary [3], [16], [17], [20].

D*-Lite [3] uses LPA* to quickly replan in a partially-unknown terrain. Anytime Dynamic A* [21] finds a suboptimal solution quickly using an inflated heuristic and then replans based on LPA* with a decreasing inflation factor. RRT# [16], LBT-RRT [20] and BIT* [17] use ideas similar to LPA* to update the (near-) optimal plan efficiently using a sequence of graphs with increasing density based on the previous search tree. ABIT* [22] uses the ideas of Truncated-LPA* [10] to truncate the inconsistency propagation of LPA* as soon as the current solution is guaranteed to be bounded suboptimal. This reduces the graph operations further. These methods have been shown to be very effective for planning in uncertain or complex environments.

In this paper, we extend COA* and propose the Class-Ordered LPA* (COLPA*) for lifelong planning in dynamic weighted colored graphs to find the shortest path among the set of paths with minimal inclusion of inferior edges. The main observation is that the regular numerical ordering of LPA* to prioritize the repairing of inconsistent sub-paths can be generalized to an abstract total order. Conveniently, the proposed COLPA* algorithm inherits all the theoretical properties of LPA* in terms of efficiency and optimality, but as in the case of COA* the optimality is with respect to the total order on the set of paths.

II. Problem Formulation

In this section, we introduce some notation and the necessary assumptions that will be used through the rest of this paper.

A. Weighted Colored Graph

Let $G = (V, E)$ be a graph with vertex set V and edge set E. Let $\phi_V : V \to \mathcal{L}$ be a *vertex perception function* that classifies each vertex $v \in V$ to an element in a finite integer set $\mathcal{L} = \{1, ..., L\}$, such that ϕ_V partitions the set V, namely, $V = \bigcup_{\ell \in \mathcal{L}} V_\ell$ where $V_\ell = \{v \in V : \phi_V(v) = \ell\}$ and $V_i \cap V_j = \varnothing$ for $i \neq j$. Similarly, define $\phi_E : E \to \mathcal{K}$ to be an *edge perception function* that classifies each edge $e \in E$ to an element in a finite integer set $\mathcal{K} = \{1, ..., K\}$, such that $E = \bigcup_{k \in \mathcal{K}} E_k$ where each $E_k = \{e \in E : \phi_E(e) = k\}$ and $E_i \cap E_j = \varnothing$ for $i \neq j$. We will assume that the edge class set is larger than the vertex class set, i.e., $L \leq K$, and that, for each edge, $e = (u, v)$, it holds that $\phi_E(e) \geq \max\{\phi_V(u), \phi_V(v)\}$. This assumption allows us to quickly underestimate the edge class by classifying the end vertices first. Also, for each edge $e \in E$, a weight function $w : E \to \mathbb{R}_+$ assigns a non-negative real number, e.g., the distance to traverse this edge. We say that an edge e is evaluated when both the values $\phi_E(e)$ and $w(e)$ are computed.

B. Optimal Paths

Define a path $\pi = (v_1, v_2, \ldots, v_m)$ on the graph $G = (V, E)$ as an ordered set of distinct vertices $v_i \in V$, $i = 1, \ldots, m$, such that for any two consecutive vertices v_i, v_{i+1}, there exists an edge $e = (v_i, v_{i+1}) \in E$. We will interchangeably denote a path as the set of such edges throughout this paper. Let $v_\mathrm{s}, v_\mathrm{g} \in V$ be the start and goal vertices, respectively. Denote by $\Pi(v_\mathrm{s}, v)$ (or $\Pi(v)$ in short when there is no danger of ambiguity) the set of all paths from v_s to some vertex v in G. Let $\Pi_k(v)$ be the subset of $\Pi(v)$ in which the worst (greatest) edge class included in each path in $\Pi_k(v)$ is exactly k. That is,

$$\Pi_k(v) = \{\pi \in \Pi(v) : d(\pi, k) > 0 \text{ and } d(\pi, \ell) = 0, \forall \ell > k\},$$

where $d(\pi, k) = \sum_{\{e \in \pi : \phi_E(e) = k\}} w(e)$ is the sum of edge weights that are of class k in path π. Furthermore, define

$$\Pi_k^i(v) = \{\pi \in \Pi_k(v) : d(\pi, k) = i\},$$

to be the set of class-k paths which have exactly i-long edges of class k. We shall impose a total order on the set of paths from v_s to any vertex v with $\Pi_k^i(v) \prec \Pi_\ell^j(v)$ for any i, j whenever $k < \ell$ and $\Pi_k^i(v) \prec \Pi_k^j(v)$ for all $i < j$. Hence, we define the optimal path set $\Pi^*(v)$ as the nonempty set of paths to v having best worst-class edge with the shortest worst-class edges, that is,

$$\Pi^*(v) = \min_{i \in \mathbb{R}} \min_{k \in \mathcal{K}} \{\Pi_k^i(v)\},$$

where the minimum is defined with respect to the total path order. The optimal path problem is to find the path π^* from v_s to v_g which is the shortest path in $\Pi^*(v_\mathrm{g})$, that is,

$$\pi^* = \operatorname*{argmin}_{\pi \in \Pi^*(v_\mathrm{g})} \sum_{e \in \pi} w(e).$$

III. The COLPA* Algorithm

Before we describe our algorithm, we start by detailing the subtle-yet-important different data structures it uses, namely, the search tree (Section III-A) and the priority queue (Section III-B). We then show (Section III-C) how these are used to define COLPA*.

A. Search Tree

In typical search algorithms, such as A*, each path stores the so-called cost-to-come, namely, the accumulated cost along the path edges. However, as we will see, in our setting it will be beneficial to store the accumulated cost for each color individually. Thus, in our setting, the values of cost-to-come are not real numbers but vectors in $\mathbb{R}^{|\mathcal{K}|}$, where the k-th index stores $d(\pi, k)$, the sum of edge weights along the path π whose class is k. We denote this vector value by $\theta(\pi) = (d(\pi, k))_{k \in \mathcal{K}} \in \mathbb{R}^{|\mathcal{K}|}$. Note that the total order defined over two paths π_1 and π_2 from the same start vertex to the same goal vertex induces a total order on the vector values, that is, $\theta(\pi_1) \prec \theta(\pi_2)$, if $\pi_1 \prec \pi_2$. The other direction is also true, when a pair of cost-to-come values for the paths from the same start and goal vertices are compared. That is, $\theta(\pi_1) \prec \theta(\pi_2)$ implies $\pi_1 \prec \pi_2$ if both π_1 and π_2 have the same goal vertices.

Each node in the search tree that COLPA* constructs corresponds to a path to a given vertex. It stores two cost-to-come values for this vertex, namely, the g-value and the right-hand-side value or rhs-value [9]. The g-value is the accumulated cost-to-come by traversing the previous search tree, whereas the rhs-value is defined as

$$rhs(v) := \begin{cases} 0, & \text{if } v = v_\mathrm{s}, \\ \min_{u \in pred(v)} \{g(u) + \theta(u, v)\}, & \text{otherwise.} \end{cases}$$

Here the addition is a vector addition in $\mathbb{R}^{|\mathcal{K}|}$, the minimum is defined with respect to the path-induced total order, and $pred(v)$ is the predecessors of the vertex v in the graph G. Hence, the rhs-value is an one-step better informed estimate of the cost-to-come than the g-value.

Additionally, each node stores a backpointer for the corresponding vertex. The backpointer of a vertex v is the predecessor vertex which minimizes the rhs-value of v, and it is denoted by

$$bp(v) := \begin{cases} \varnothing, & \text{if } v = v_\mathrm{s}, \\ \operatorname{argmin}_{u \in pred(v)} \{g(u) + \theta(u, v)\}, & \text{otherwise.} \end{cases}$$

Hence, the path to v is easily retreived by following the backpointers of v to the start vertex v_s.

A vertex v is called overconsistent if $rhs(v) \prec g(v)$ and underconsistent if $g(v) \prec rhs(v)$. Otherwise, a vertex v is consistent, in which case we write $rhs(v) = g(v)$. In other words, a vertex v is overconsistent if the cost-to-come of this vertex was overestimated previously, and v is underconsistent if the cost-to-come of this vertex was underestimated previously.

B. Priority Queue

We use an edge queue Q to prioritize the expansion (i.e., evaluation) of inconsistent vertices in the same way LPA* sorts the inconsistent vertices, except the key component values are not real numbers but they are vectors. We assume that there exists a consistent heuristic cost-to-go function $h : V \to \mathbb{R}^{|\mathcal{K}|}$ which assigns to a vertex an admissible estimate cost-to-go to the goal vertex v_g with $h(u) \preceq \theta(u, v) + h(v)$ for any $u \in V$ and $v \in succ(u)$, the successors of u. A trivial consistent heuristic function maps to $0 \in \mathbb{R}^{|\mathcal{K}|}$.

The key $k(v)$ of vertex v contains two components : $k(v) := [k_1(v); k_2(v)]$, where $k_1(v) = \min\{g(v), rhs(v)\} + h(v)$ and $k_2(v) = \min\{g(v), rhs(v)\}$. Keys are sorted with lexicographic ordering, that is $k(v) \prec k'(v)$ if and only if either $k_1(v) \prec k_1'(v)$ or $(k_1(v) = k_1'(v)$ and $k_2(v) \prec k_2'(v))$.

C. Details of the Algorithm and Main Procedures

The Class-Ordered LPA*(COLPA*) is similar to the regular LPA*, except that COLPA* obeys the total order hereto defined in the vector space $\mathbb{R}^{|\mathcal{K}|}$. All the vertices of the graph G are implicitly initialized with the cost-to-come values set to infinity. Also, we denote $g(v) = rhs(v)$ if and only if two vectors are equal componentwise. The rest of the algorithm is identical to LPA* [9], but for completness of the discussion, we give a brief description of the algorithm.

Algorithm 1 Class-Ordered LPA*(G, v_s, v_g)

```
1:  procedure CalculateKey(v) return
2:      [min{g(v), rhs(v)} + h(v) ; min{g(v), rhs(v)}];
3:  procedure UpdateVertex(v)
4:      if v ≠ v_s then
5:          bp(v) = argmin_{u∈pred(v)}(g(u) + θ(u, v));
6:          rhs(v) = g(bp(v)) + θ(bp(v), v);
7:      if v ∈ Q then Q.Remove(v);
8:      if g(v) ≠ rhs(v) then
9:          Q.Insert((v, CalculateKey(v)));
10: procedure ComputeShortestPath()
11:     while Q.TopKey ≺ CalculateKey(v_g) or
12: g(v_g) ≠ rhs(v_g) do
13:         u = Q.Pop();
14:         if rhs(u) ≺ g(u) then
15:             g(u) = rhs(u);
16:             for all v ∈ succ(u) do UpdateVertex(v);
17:         else
18:             g(u) = ∞;
19:             for all v ∈ succ(u) ∪ {u} do
20:                 UpdateVertex(v);
21: procedure Main()
22:     Q ← ∅;
23:     rhs(v_s) = 0;
24:     UpdateVertex(v_s);
25:     while true do
26:         ComputeShortestPath( );
27:         Wait for changes in E;
28:         L ←the set of edges that changed;
29:         for all e = (u, v) ∈ L do
30:             UpdateVertex(v);
```

We begin the search by initiating a search tree T with the start vertex as the root by setting $rhs(v_s) = 0$. Then, we put the start vertex in the priority queue by updating this vertex. The operation UPDATEVERTEX(v) finds a new optimal parent for a vertex v, and then puts this vertex according to its key values if the cost-to-come of the path up to this vertex has become inconsistent. Since all vertices are initialized with an infinite cost-to-come, all the vertices that are updated during the first search with a finite cost-to-come become inconsistent, and consequently they will be put in the priority queue. This makes LPA*/COLPA* equivalent to A*/COA* [14] during the first search.

In the main search loop, COMPUTESHORTESTPATH() removes the top vertex from the priority queue, and then expands it. The expansion of a vertex consists of the following procedures. If the top vertex in the priority queue is overconsistent, then the algorithm makes this vertex consistent by assigning the g-value of this vertex to its rhs-value. Then, the algorithm updates all the successors to propagate the inconsistencies that may have arisen due to this assignment. If the top vertex of the priority queue is underconsistent, then the algorithm makes this vertex either overconsistent or consistent (if it was already infinity) by assinging a value of infinity. Then, the algorithm updates this vertex and all its successors. Note that the g-value of a vertex can only be changed during the COMPUTESHORTESTPATH operation when the vertex is expanded.

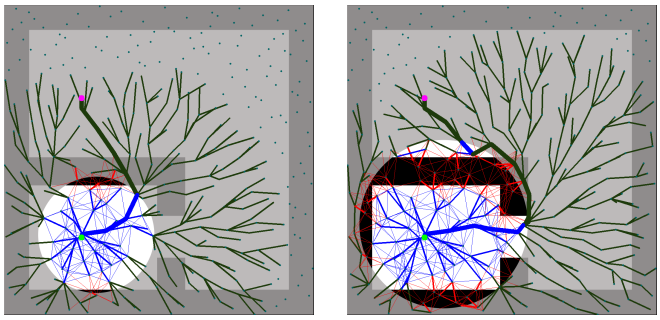
Since the operation UPDATEVERTEX puts the vertex in the priority queue only if the vertex is inconsistent, the priority queue contains only the inconsistent vertices. The priority queue prioritizes the expansion of inconsistent vertices in the same manner the A* algorithm prioritizes the expansion of optimal path candidates, and hence the key values of the expanded vertices is monotonically non-decreasing within the COMPUTESHORTESTPATH operation. Hence, when the goal vertex becomes consistent, or there does not exist any inconsistent vertices that could possibly improve the goal cost-to-come, then the current path to the goal is the optimal path, terminating the search. The algorithm then waits for any graph changes. When there are changes in the graph, either geometrically or semantically in any of the edges of the graph, the algorithm updates the end vertices of these changed edges in order to propagate the repairing of any relevant inconsistencies accordingly.

COLPA* inherits the theoretical properties of LPA* in terms of optimality and efficiency. That is, when the COMPUTESHORTESTPATH terminates, the path from v_s to v_g obtained by following the backpointers is no worse than π^*, the shortest path among the set of paths with minimal inclusion of inferior edges. Also, during the COMPUTESHORTESTPATH operation, no vertices are expanded more than twice. Note that when the graph is unicolored, COLPA* reduces to LPA*. In that case, the same number of vertices will be expanded in the same order.

Figure 2 shows the search instances of COLPA* in a partially-known 2D environment after the COMPUTESHORTESTPATH operation terminates, where new sensor readings update the map. A graph is constructed by sampling vertices using a Halton sequence [23], and then the same graph is used throughout without additions or deletions of vertices and edges. When a vertex v is updated during the UPDATEVERTEX(v) procedure, each of the neighboring edges is evaluated, that is, the value of $\theta(u, v)$ for all $u \in pred(v)$ is computed. All the evaluated edges in this example belong to a class in $\mathcal{K} = \{\text{feasible}, \text{unknown}, \text{infeasible}\}$. COLPA* always finds the shortest path with minimal inclusion of the edges in collision (red), then the edges that are unknown (gray), then the feasible edges (blue). In our implementation we used a straight-line path from a vertex to the goal vertex as the heuristic cost-to-go of that vertex.

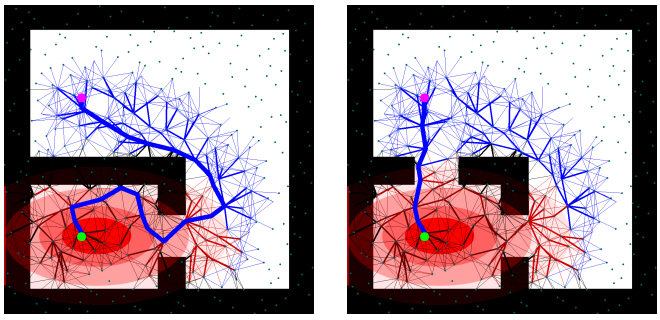
In another example, depicted in Fig. 3, COLPA* finds the optimal solution in a dynamic environment where a certain region of the environment is known to be hazardous. The edges of the graph can be classified into six colors depending on the intensity level. COLPA* finds the shortest path among the set of paths that include shortest edges in the hazardous region, in the order of high intensity to low intensity.

Note that COLPA* finds the optimal solution that COA* would find it in the same environment. The differences



(a) First Search (b) Second Search

Fig. 2: COLPA* search on Halton-sequence graphs in a partially-known environment, where new sensor reading updates the map over time. Blue, gray and red are feasible, unknown, and infeasible edges respectively. Light, semi-bold, and bold edges are, respectively, the evaluated edges, the edges in the current search tree, and the edges in the solution path from ● to ●.



(a) First Search (b) Second Search

Fig. 3: COLPA* search on Halton sequence graphs in a dynamic environment with known hazardous region (red). COLPA* finds the shortest path among the set of paths which have the shortest edges in the hazardous region.

between the two algorithms are that COLPA* uses the search tree constructed in the previous iteration to selectively repair only the relevant inconsistent part of the current search, whereas COA* builds a new search tree from scratch. Also, COLPA* evaluates all the incident edges when it expands a vertex, whereas COA* uses an edge queue with heuristically-estimated edge values to lazily construct the search tree. This results in fewer edge evaluations at the expense of more vertex expansions [24], [25].

IV. Numerical Examples

We compared the solution cost and the number of vertex expansions required to find the optimal solution for both of the COLPA* and LPA* algorithms in a dynamic environment. Note that COLPA* on a unicolor weighted graph is equivalent to LPA*, and hence the same number of vertices will be expanded and the solution cost will be also the same. The solutions of both algorithms become qualitatively different only when the graph has more than one color. We show some examples where the semantic information of the environment is utilized by COLPA* to find a solution faster compared to LPA* by reducing the relevant search space.

Suppose we have some prior knowledge about the environment that some region of the environment contains the shortest path from a point A to point B. Then, coloring that region with high-level class will make the rest of the region one class inferior. Hence, COLPA* searches in a reduced search space, expanding a fewer number of vertices to find the optimal solution. Figure 4 shows the search instances of COLPA* with two different semantic guidance in comparison to LPA*. The top row shows the search trees constructed by COLPA* over three consecutive search episodes in a dynamic environment without any prior knoweldge, which is equivalent to LPA*. The middle and the bottom rows show the search trees of COLPA* in the same environment with two different priors, colored in green and blue, respectively.

We denote by COLPA*-G, the version of COLPA* which considers the green region as the top class, hence making the other regions relatively one class inferior. The optimal path is the shortest path among the set of paths that has the shortest edges in collision, then outside of the green region, and then inside the green region. Likewise, COLPA*-B is the version of COLPA* that takes the blue region to be the top class, and replans for the shortest path within the blue region. The blue region is the subset of the green region containining the optimal solution. Hence, COLPA*-B exploits better information than COLPA*-G, resulting in a fewer number of verex expansions. The number of vertices expanded for each search episode, and the solution length are shown in Table I (First Experiment).

In the second experiment, we consider two different priors for a dynamic environment: one with an "aggressive"' prior that guides the search close to obstacles, and the other with a "conservative" prior which leaves a larger clearance from the obstacles. These priors are colored in green and blue respectively, and we denote the version of COLPA* with the aggressive prior COLPA*-G and the version of COLPA* with the conservative piror COLPA*-B. The aggressive prior allows COLPA* to find shorter paths than the conservative prior, while the conservative prior yields longer paths which need not be replanned upon environment changes. This is depicted in Figure 5. The number of vertices expanded for each search episode and the solution lengths are shown in Table I (Second Experiment).

Note that if there exists a solution within the top class region, then COLPA* will find the shortest path within this top class region. If no solution exists within this region, then COLPA* will still try to find the shortest path in the next best set of paths.

V. Conclusion

Replanning is crucial for many robotics applications, where the environment is dynamic or too complex to be captured in a single model. Consequently, when the model of the environment changes, the plan needs to be updated accordingly. Incremental-search methods efficiently use previous search results to facilitate a new plan when the model

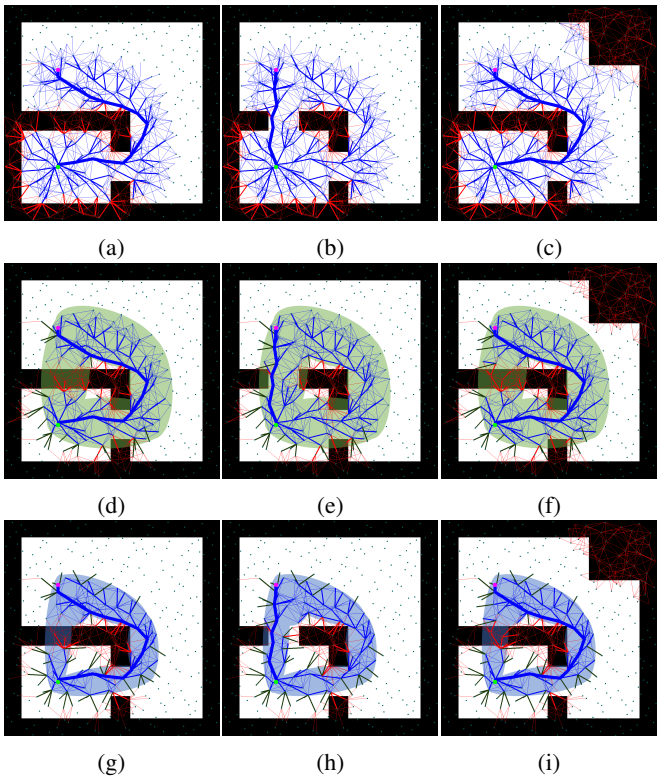


Fig. 4: LPA* (top), COLPA*-G (middle), and COLPA*-B (bottom) search on a dynamic environment from ● to ●, where COLPA*-G favors the green region during the search, and COLPA*-B favors the blue region during the search. The green region contains the blue region, and the blue region contains the optimal solution.

changes, and yet they rely solely on geometric information of the environment that is encoded as numerical weights on the edges of the graph. Semantic information of the environment, on the other hand, can provide crucial insight for good plans.

We have extended LPA* to operate on weighted colored graphs, by generalizing the numerical order of the regular LPA* algorithm to the total order defined on the set of paths. The proposed Class-Ordered LPA* (COLPA*) algorithm finds the shortest path among the set of paths with minimal inclusion of inferior colors. COLPA* inherits the theoretical properties of LPA*, namely, the optimality and efficiency, except that the notion of optamality is based on the total order on the set of paths with colored edges.

Our numerical examples illustrate that semantic information can benefit planners to find the shortest path faster. This is because COLPA* expands a fewer number of vertices as the semantic information effectively reduces the relevant region of the search space.

Finally, it should be pointed out hat COLPA* can also be used to guide a robot's search by a human operator. This is similar in nature to the work by Ranganeni et al. [26] who showed that rough human guidance can be used to speed up a planner's capabilities. Alternatively, this information can be learned by the robot from previous experiences and used to

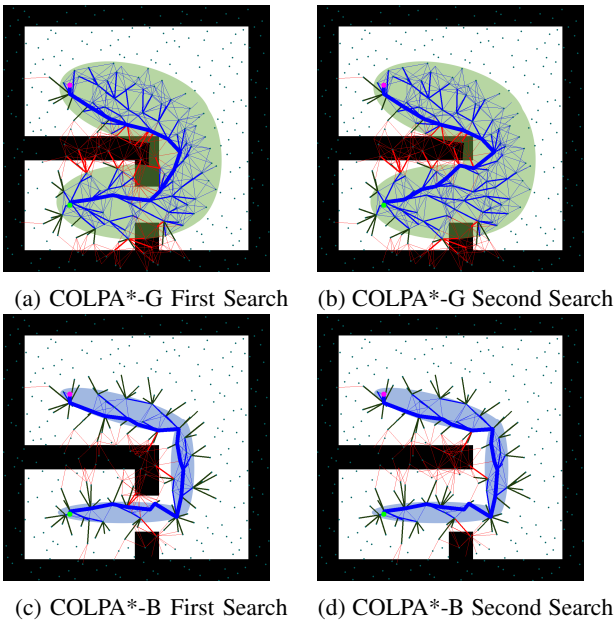


Fig. 5: COLPA* search with two different priors : aggressive (green) and conservative (blue). COLPA*-G finds shorter paths than COLPA*-B, while COLPA*-B does not replan when the environment changes.

search in relevant regions of the environment first. Obtaining such good semantic information about the environment is a good topic for future investigation.

Acknowledgement

This work has been supported by ARL under DCIST CRA W911NF-17-2-0181 and NSF under award IIS-2008686.

References

[1] L. Janson, T. Hu, and M. Pavone, "Safe motion planning in unknown environments: Optimality benchmarks and tractable policies," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, PA, June 26–30 2018.

[2] A. T. Stentz, "The focussed D* algorithm for real-time replanning," in *Proceedings of 14th International Joint Conference on Artificial Intelligence*, Montreal, Canada, August 20–25 1995, p. 1652 – 1659.

[3] S. Koenig and M. Likhachev, "D* lite," in *18th National Conference on Artificial Intelligence*, Edmonton, Canada, July 28–Aug 1 2002, p. 476–483.

[4] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354–363, 2005.

[5] M. Likhachev and A. Stentz, "PPCP: Efficient probabilistic planning with clear preferences in partially-known enviornments," in *AAAI*, Boston, MA, July 16–20 2006, p. 860–867.

[6] V. R. Desaraju and J. P. How, "Decentralized path planning for multi-agent teams in complex environments using rapidly-exploring random trees," in *IEEE International Conference on Robotics and Automation*, Shanghai, China, May 9–13 2011, pp. 4956–4961.

[7] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[8] G. Ramalingam and T. Reps, "An incremental algorithm for a generalization of the shortest-path problem," *Journal of Algorithms*, vol. 21, pp. 267–305, 1996.

[9] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A*," *Artificial Intelligence*, vol. 155, no. 1, pp. 93 – 146, 2004.

[10] S. Aine and M. Likhachev, "Truncated incremental search," *Artificial Intelligence*, vol. 234, pp. 49 – 77, 2016.

TABLE I: Number of vertex evaluations and solution length for LPA* and COLPA* with different semantic information recorded over consecutive search queries in a dynamic environment.

First Experiment	LPA*	COLPA*-G	COLPA*-B
First Query			
# Vertex Expansion	91	67	54
Solution Length	1.039	1.039	1.039
Second Query			
# Vertex Expansion	7	7	7
Solution Length	0.465	0.465	0.465
Third Query			
# Vertex Expansion	9	9	9
Solution Length	1.039	1.039	1.039
Total # V-Expansion	107	83	70
Second Experiment	LPA*	COLPA*-G	COLPA*-B
First Query			
# Vertex Expansion	91	63	41
Solution Length	1.039	1.039	1.232
Second Query			
# Vertex Expansion	29	29	0
Solution Length	0.996	0.996	1.232
Total # V-Expansion	120	92	41

[11] D. Wooden and M. Egerstedt, "On finding globally optimal paths through weighted colored graphs," in *Proceedings of the 45th IEEE Conference on Decision and Control*, San Diego, CA, December 13–15 2006, pp. 1948–1953.

[12] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[21] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," *Artificial Intelligence*, vol. 172, no. 14, pp. 1613 – 1643, 2008.

[13] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, July 1968.

[14] J. Lim and P. Tsiotras, "A Generalized A* Algorithm for Finding Globally Optimal Paths in Weighted Colored Graphs," *arXiv e-prints*, December 2020.

[15] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[16] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *IEEE International Conference on Robotics and Automation*, Karlsrühe, Germany, May 6–10 2013, pp. 2421–2428.

[17] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *IEEE International Conference on Robotics and Automation*, Seattle, WA, May 26–30 2015, pp. 3067–3074.

[18] M. P. Strub and J. D. Gammell, "Adaptively informed trees (AIT*): Fast asymptotically optimal path planning through adaptive heuristics," in *IEEE International Conference on Robotics and Automation*, May 31–Aug 31 2020, pp. 3191–3198.

[19] O. Salzman and D. Halperin, "Asymptotically-optimal motion planning using lower bounds on cost," in *IEEE International Conference on Robotics and Automation*, Seattle, WA, USA, 26-30 May 2015, pp. 4167–4172.

[20] ——, "Asymptotically near-optimal RRT for fast, high-quality motion planning," *IEEE Trans. Robotics*, vol. 32, no. 3, pp. 473–483.

[22] M. P. Strub and J. D. Gammell, "Advanced BIT* (ABIT*): Sampling-based planning with advanced graph-search techniques," in *IEEE International Conference on Robotics and Automation*, Paris, France, May 31–Aug 31 2020, pp. 130–136.

[23] J. H. Halton, "Algorithm 247: Radical-inverse quasi-random point sequence," *Communications of the ACM*, vol. 7, no. 12, pp. 701–702, December 1964.

[24] B. Cohen, M. Phillips, and M. Likhachev, "Planning single-arm manipulations with n-arm robots," in *Proceedings of Robotics: Science and Systems*, Berkeley, CA, July 12–16 2014.

[25] A. Mandalika, O. Salzman, and S. Srinivasa, "Lazy receding horizon A* for efficient path planning in graphs with expensive-to-evaluate edges," in *Proceedings of the International Conference on Automated Planning and Scheduling*, Delft, Netherlands, 2018, pp. 476–484.

[26] V. Ranganeni, S. Chintalapudi, O. Salzman, and M. Likhachev, "Effective footstep planning using homotopy-class guidance," *Artif. Intell.*, vol. 286, p. 103346, 2020.